Knowledge-based Consistency Testing of Large Language Models

Sai Sathiesh Rajan[*1], Ezekiel Soremekun[†2], and Sudipta Chattopadhyay[‡1]

[1]Singapore University of Technology and Design, Singapore
[2]Royal Holloway, University of London, UK

Abstract

In this work, we systematically expose and measure the *inconsistency* and *knowledge gaps* of Large Language Models (LLMs). Specifically, we propose an automated testing framework (called KONTEST) which leverages a *knowledge graph* to construct test cases. KONTEST probes and measures the inconsistencies in the LLM's knowledge of the world via a combination of semantically-equivalent queries and test oracles (metamorphic or ontological oracle). KONTEST further mitigates knowledge gaps via a weighted LLM model ensemble. Using four state-of-the-art LLMs (`Falcon`, `Gemini`, `GPT3.5`, and `Llama2`), we show that KONTEST generates 19.2% error inducing inputs (1917 errors from 9979 test inputs). It also reveals a 16.5% knowledge gap across all tested LLMs. A mitigation method informed by KONTEST's test suite reduces LLM knowledge gap by 32.48%. Our ablation study further shows that `GPT3.5` is not suitable for knowledge-based consistency testing because it is only 60%-68% effective in knowledge construction.

1 Introduction

Large language models (LLMs) are being increasingly utilized in real-world applications. LLMs are powerful in solving many tasks, but their *reliability* remains a concern (Qiu et al., 2023). This is alarming because inconsistent behaviors adversely affect critical downstream tasks and influence adoption.

In this paper, we study the problem of assessing inconsistency in LLM behaviors. Previous works have demonstrated the prevalence and severity of *inconsistent* responses in LLMs (Min et al., 2023; Berglund et al., 2023; Sallou et al., 2023). To address this challenge, we conceptualize and design KONTEST[1] – a novel *test generation methodology to systematically discover consistency errors in LLMs and highlight their knowledge gaps*.

Figure 1 shows examples of consistency errors and knowledge gap discovered by KONTEST in `GPT3.5`. These are errors in `GPT3.5` about the `place` (spatial) domain. These errors may have adverse effects in critical areas (e.g., automobiles, aeronautics) where spatial LLMs are deployed, e.g., navigation systems – MapBox's MapGPT (Mapbox), LLM-Geo (Li and Ning, 2023) and MapGPT (Chen et al., 2024), L3MVN (Yu et al., 2023)). KONTEST allows to automatically discover and expose such errors to end-users/developers. This is the *first* step to enable their mitigation and repair for LLM improvement.

KONTEST leverages a *knowledge graph* for consistency testing (see Figure 2). It first automatically extracts a set of entities and entity relationships from the knowledge graph. This is then used to systematically generate (semantically-equivalent) *yes/no* queries and create test cases for the *subject LLM under test* (SUT) across various settings (e.g., atomic query vs. sequential query). The test case generation involves minimal, one-time effort for each type of entity relation considered (e.g., only two templates for 6730 test cases) and such templates can be reused for testing arbitrary LLMs. KONTEST's test suite can additionally be used to mitigate knowledge gaps by leveraging the likelihood of LLM inconsistencies to construct a weighted model ensemble. To the best of our knowledge, KONTEST *is the first systematic approach for automatically generating consistency tests for assessing LLMs*.

Knowledge-based test generation provides a unique and systematic method for exploring the SUT's knowledge and compute a *test adequacy metric* for the SUT in terms of the covered entities and relations. More importantly, the responses from the SUT allow KONTEST to construct the *SUT's knowledge base* as a subset of the extracted

[*]sai_rajan@mymail.sutd.edu.sg
[†]ezekiel.soremekun@rhul.ac.uk
[‡]sudipta_chattopadhyay@sutd.edu.sg

[1]KONTEST means **K**nowledge-based **CON**sistency **TEST**ing

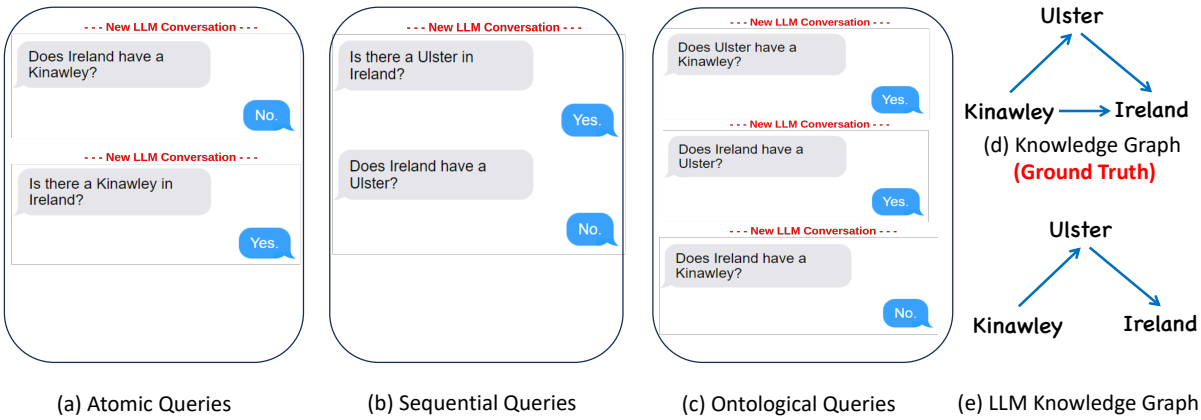


Figure 1: Examples of queries generated and errors discovered by KONTEST in `GPT3.5`: Error reflected (a) in two different conversations asking semantically equivalent questions (**Atomic** error in Table 1) and (b) in conversation asking semantically equivalent questions in a sequence (**Sequential** error in Table 1). (c) Example of (**Ontological**) error in Table 1, (d) the knowledge graph extracted for the considered entities i.e., Ulster, Kinawley and Ireland, (e) SUT's knowledge showing that the LLM lacks the knowledge about *Kinawley is in Ireland*.

knowledge graph and to concretely highlight the knowledge gap of the SUT. This is valuable information for developers, it enables refining the model and improving its knowledge.

KONTEST differs from existing works both in its objective and in its unique approach for testing LLMs. Prior works have focused on other non-functional properties such as reasoning ability, robustness and security (Honarvar et al., 2023; Wu et al., 2023; Yang et al., 2023b). Few works have demonstrated the importance of LLM consistency, e.g., via self-consistency (Min et al., 2023) and reversal curse (Berglund et al., 2023). In contrast, KONTEST employs *automated test generation* to discover a larger scope of consistency errors.

This paper provides an overview of KONTEST (section 2), and makes the following contributions:

1. We present KONTEST, a novel approach to leverage *knowledge graph* for checking the *consistency* of LLM responses and measure their *knowledge gaps* (section 3).
2. We present the *metamorphic* and *ontological* oracles that allow to check consistency errors in LLM results (section 3).
3. We implement KONTEST and evaluate it with `Falcon` (Nomic AI, 2023), `Gemini` (Anil et al., 2023), `GPT3.5` (OpenAI, 2023), and `Llama2` (Touvron et al., 2023). KONTEST revealed 19.2% erroneous inputs and exposed an average knowledge gap of 16.5% (section 5).
4. We propose a technique that mitigates knowledge gaps via a weighted model ensemble via the likelihood of LLM inconsistencies obtained from KONTEST's test suite. The mitigation technique reduces knowledge gaps by 32.48% (section 5).
5. We perform an ablation study by replacing each component of KONTEST with `GPT3.5` using few-shot prompting. We found that `GPT3.5` constructs at most 68% of the ground truth knowledge base. It exhibits up to 63.7% false positives in error detection (section 5).

After the related works (section 6), we conclude in section 7 and discuss limitations (section 8).

2 Overview

In this section, we outline the motivation behind our approach and present an illustrative example to demonstrate the overall process of our approach.

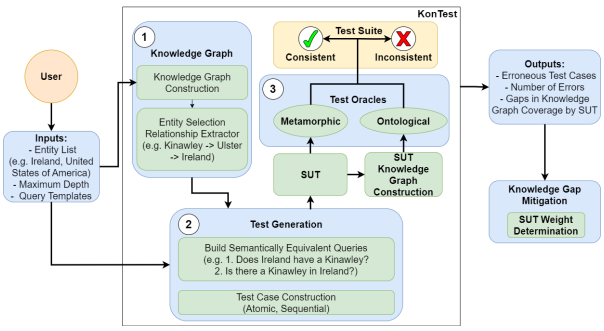
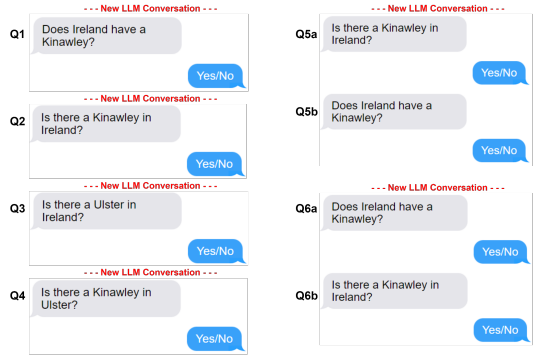


Figure 2: Overall workflow of our approach (KONTEST)

Key Insight: The key insight behind KONTEST is to leverage a knowledge graph for systematic consistency testing of LLMs. The knowledge graph

Table 1: Test cases generated and error types detected by KONTEST using metamorphic and ontological oracles.

Generated Sentences	Error Type	Oracle
	`atomic`	LLM(Q1)=LLM(Q2)
	`sequential-intra`	LLM(Q5a)=LLM(Q5b), LLM(Q6a)=LLM(Q6b)
	`sequential-inter`	LLM(Q1)=LLM(Q5b), LLM(Q2)=LLM(Q6b)
	`Ontological`	(LLM(Q3)=LLM(Q4)=**Yes**) → (LLM(Q2)=**Yes**)

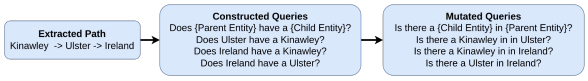


Figure 3: Example queries and templates for the "Places" domain in KONTEST

serves multiple purposes in KONTEST: Firstly, entities and entity relations extracted from the knowledge graph allows KONTEST to systematically generate queries and construct test cases for validating the consistency of the subject LLM (SUT). The LLM's responses to these test cases allow KONTEST to build the knowledge (sub)-graph where the LLM behaves correctly. Secondly, knowledge-based test generation allows KONTEST to report a *test adequacy* metric for the SUT in terms of the entities and relations covered within the knowledge graph. Finally, this approach enables KONTEST to highlight sub-graph of the knowledge graph where the outputs from the SUT are logically *inconsistent*. This enables practitioners to selectively focus on the knowledge gaps highlighted by KONTEST for improving the LLM (e.g., by fine-tuning).

Running Example: Figure 2 outlines our approach. KONTEST broadly comprises of three key components: ① knowledge graph construction ("Knowledge Graph" in Figure 2), ② test generation ("Test Generation" in Figure 2), and ③ test oracles ("Test Oracles" in Figure 2). Given a set of entities (e.g., countries) and a maximum graph depth, KONTEST locates the given entities in the knowledge graph (e.g., Wikidata knowledge graph (Vrandečić and Krötzsch, 2014)), and constructs the associated knowledge graph before extracting the relevant entities and relationships. Given the relationships present in Figure 1(d), KONTEST constructs two semantically equivalent queries for each relation with the aid of a template (see Figure 3) relating the two involved entities. The resultant queries are then fed to the SUT generating responses to both atomic and sequential queries. Figure 1(a) shows two such semantically equivalent queries, each of which was part of a new conversation with `GPT3.5` (aka "`atomic`"), while Figure 1(b) illustrates one such consistency error with inconsistent responses for semantically-equivalent queries that were within the same conversation (aka "`sequential-inter`"). Finally, Figure 1(c) illustrates a series of queries, whose responses from `GPT3.5` collectively show inconsistent behavior (aka "`ontological`"). Specifically, the positive responses to the first two queries imply that *Ireland does have a Kinawley*. However, due to the negative response to the third query, our ontological oracle reveals a consistency error. We further note that such errors cannot be uncovered by counter-questioning LLMs. The relevant sub-graph of the knowledge graph for `GPT3.5` is shown in Figure 1(e). This clearly illustrates the knowledge gap (i.e., *Kinawley*↛*Ireland*) for `GPT3.5`.

3 Methodology

3.1 Knowledge Graph based Test Generation

***Knowledge Graph Construction*:** KONTEST allows the developer to specify the list of entities that are of interest. With these initial set of entities, KONTEST then queries an external source of knowledge to compute, up to a certain *depth*, all other related entities for the considered relation. Developers can also control this *depth*, allowing them to determine the degree to which the vicinity of the initial set of entities is explored. Once the initial knowledge base is constructed, KONTEST

Algorithm 1 KG-Based Query Generation

```
1: procedure GEN_TEST(𝕂𝔾)
2:     Query_list, MutQuery_list ← ∅
3:     for 𝕂𝔾_node1 in 𝕂𝔾 do
4:         for 𝕂𝔾_node2 in 𝕂𝔾 do
5:             if 𝕂𝔾_node1 ≠ 𝕂𝔾_node2 then
6:                 ▷ Finds relationship between 𝕂𝔾_node1 and 𝕂𝔾_node2
7:                 𝕂𝔾_rel ← Find_Relation(𝕂𝔾_node1, 𝕂𝔾_node2)
8:                 ▷ Builds query using the relation (𝕂𝔾_node1, 𝕂𝔾_node2)
9:                 Query ← Gen_Query(𝕂𝔾_node1, 𝕂𝔾_node2, 𝕂𝔾_rel)
10:                Query_list ← Query_list ∪ {Query}
11:                ▷ Generates semantically equivalent query
12:                Query_M ← Mut_Gen(Query)
13:                MutQuery_list ← MutQuery_list ∪ {Query_M}
14:    return Query_list, MutQuery_list
```

Algorithm 2 Graph Checker

```
1: procedure GRAPH_CHECKER(Graph_Gen)
2:     Err_count, Coverage_count ← ϕ
3:     for Node_1 in Graph_Gen do
4:         for Node_2 in Graph_Gen do
5:             if Node_1 ≠ Node_2 then
6:                 ▷ Checks whether direct path exists between nodes.
7:                 Path_Dir ← Dir_Path(Node_1, Node_2)
8:                 if Path_Dir ≠ TRUE then
9:                     ▷ Checks whether indirect path exists between nodes.
10:                    Path_InDir ← InDir_Path(Node_1, Node_2)
11:                    if Path_InDir == TRUE then
12:                        Err_count ← Err_count + 1
13:                else
14:                    Coverage_count ← Coverage_count + 1
15:    return Err_count, Coverage_count
```

extracts the full path, $\mathbb{KG}$, associated with a leaf node of the knowledge base. This path is then used to guide the test generation process. For example, given the leaf entity *Kinawley*, as shown in Figure 1, KONTEST extracts the $\mathbb{KG}$ for *Kinawley* as $Kinawley \rightarrow Ulster \rightarrow Ireland$.

***Test Generation*:** Given a knowledge path $\mathbb{KG}$, KONTEST leverages Algorithm 1 to exhaustively generate queries relating to all possible pairs of entities present in the path. To this end, KONTEST first finds the relation, $\mathbb{KG}_{rel}$, between a pair of nodes. KONTEST then generates a query, $Query$, using $\mathbb{KG}_{rel}$ with the aid of the template shown in Figure 3 (line 9 in Algorithm 1). KONTEST also generates the mutated query, $Query_M$, with another template (line 12 in Algorithm 1). These templates are dependent on the relation between the two nodes in question. For instance, Figure 3 shows the set of queries and mutated queries (i.e., $Query_{list}$ and $MutQuery_{list}$) generated from one particular path and along with the respective templates used to generate them.

We note that developers can implement additional templates easily if they are interested in querying different relations. Moreover, creation of these templates is a one-time process and the cost incurred is minimal (<10 LoC) since the number of different types of relationships is small in comparison to the number of entities. We further note that our templates are directly applicable to other domains with similar relations. For instance, our templates for the "Places" domain can be reused for the "Food" domain since both domains have a "contains" type relation.

3.2 Test Oracles

KONTEST detects consistency errors in LLM via a metamorphic oracle and an ontological oracle.

***Metamorphic Oracle*:** This oracle leverages both the original ($Query_{list}$) and mutated ($MutQuery_{list}$) query sets to create four unique conversations. These conversations are then fed into the target LLM (SUT). Concretely, KONTEST creates two conversations containing the answer to both (initial and mutated) `atomic` queries individually (see the first column of "Generated Sentences" in Table 1) and two conversations where the queries are fed in a `sequential` manner (see the second column of "Generated Sentences" in Table 1). Responses from these conversations are then evaluated using the consistency checker embodied in KONTEST to identify the erroneous test cases and the number of each error types (i.e., `atomic`, `sequential-intra` and `sequential-inter` in Table 1).

***Ontological Oracle*:** Firstly, this oracle builds the SUT's knowledge graph. To this end, the generated queries from KONTEST are fed into the SUT and the target responses are recorded. Subsequently, given a list of such responses, KONTEST identifies the nodes involved in each response. These nodes are then added to the SUT's knowledge graph. Next, if the query response is positive (i.e., correct), KONTEST introduces a directed edge between the nodes in the SUT's knowledge graph indicating that the respective relation exists between the two nodes being considered. It is important to note that the relations in question are not symmetric in nature, justifying the directed flavor of the edge. After constructing the SUT's knowledge graph, $Graph_{Gen}$, KONTEST checks for ontological consistency errors with the aid of Algorithm 2.

To check the consistency in SUT's knowledge graph, Algorithm 2 first inspects whether a direct path exists between any pair of nodes $(Node_1, Node_2)$ in $Graph_{Gen}$ (line 7 in Algorithm 2). In the absence of such a direct path, when an indirect path exists between the same pair of nodes (line 11 in Algorithm 2), KONTEST detects

an ontological consistency error and updates the error count, Err_{count}. Intuitively, a direct path between two nodes indicates that the SUT responded positively to a question relating the two nodes. Similarly, the presence of an indirect path indicates that SUT knows that a positive relationship exists between the nodes albeit through more than one queries. As such, the SUT can be considered to be exhibiting an *ontological inconsistency*. As an example, consider the ontological oracle illustrated in Table 1. In this case, if an indirect path exists between nodes for *Kinawley* and *Ireland* via node *Ulster* (i.e., the responses to both **Q3** and **Q4** are positives from the SUT), then KONTEST concludes that a direct path *should also exist between Kinawley and Ireland*. Hence, the absence of a positive response for the respective query (**Q2**) is indicated as an ontological error. As a byproduct of our ontological oracle, KONTEST computes the knowledge coverage $Coverage_{count}$ for the SUT.

3.3 Knowledge Gap Mitigation

We propose a weighted ensemble method that leverages the test suite generated by KONTEST to mitigate the discovered knowledge gaps. The ensemble method uses the number of metamorphic errors found for each SUT on a per relation basis to inform its scoring system. A relation that induces x errors is given a score of $5 - x$ since each relation can maximally induce five errors. The cumulative SUT specific score, $Score_i$, is then found by summing the scores for each relation. Next, the relative weight, W_i, for each SUT in the set of SUTs ($\mathbb{SUT}$), is computed via the following equation:

$$W_i = \frac{Score_i}{\sum Score_i} \forall i \in \mathbb{SUT} \quad (1)$$

We then compute the final ensemble score by first multiplying the relative weights with the results for each response and summing them. In particular, we consider positive and negative responses to be one and zero respectively. If the ensemble score is above the midpoint (0.5), the ensemble response for the relation is considered to be positive. To determine the effectiveness of our weighting, we compare it to *simple majority voting* – an ensemble that does *not* account for the likelihood of inconsistencies for each SUT (*see* RQ3).

4 Evaluation Setup

To evaluate our approach (KONTEST), we pose the following *research questions* (RQs):

- **RQ1 Effectiveness:** How *effective* is KONTEST in exposing consistency errors in LLMs? Are KONTEST results *stable*?
- **RQ2 Knowledge Coverage:** What is the SUT's *knowledge graph coverage*? How much is the SUT *knowledge gap* revealed by KONTEST?
- **RQ3 Knowledge Gap Mitigation:** How *effective* is the proposed weighted ensemble technique in improving knowledge coverage? How does it compare to a simple majority voting? Does it generalize to a new template?
- **RQ4 Ablation Study with `GPT3.5`:** Can the state-of-the-art LLM (`GPT3.5`) perform the three main sub-tasks of KONTEST, i.e., knowledge construction, test generation and error detection?

Knowledge Graph: We rely on Wikidata's knowledge graph (Vrandečić and Krötzsch, 2014) and Wikidata's SPARQL query service to extract information pertinent to the two tested domains: *places* and *music*. We choose these domains as inconsistent knowledge about *places* may have adversarial consequences in navigation use cases of LLMs (Mapbox). Concurrently, inconsistencies in the *music* domain may inaccurately capture the ownership of copyrighted materials (e.g., music albums). For the places domain, we take the list of countries ranked by (nominal) GDP per capita and select the top ten countries as our initial entity list (IMF). We then iteratively build our knowledge graph by finding "*administrative divisions*" that are "*located in the administrative territorial entity*" of the parent entity. For the music domain, we take the top 200 musical acts since 2000 as our initial entity list (Chart2000). We then find the set of "*albums*" that have the parent entity as a "*performer*". We also find the set of songs and singles that are "*part of*" the album being considered. Once the graph is fully constructed, we randomly select 50 leaf nodes for each domain and extract the full path associated with the selected leaf nodes.

Subject Programs and Query Construction: Table 2 outlines features of the LLMs (SUTs) tested by KONTEST. We ensure that temperature for each query is set to zero where possible. In addition, we provide the LLMs with an additional system command, "*Be concise as possible. Answer with a yes or no response,*", before the start of any conversation. In the event that the LLM does not support a system command, we prepend the statement to the queries before feeding it to the LLM. For

Table 2: Model Details

LLM	Lineage	Model	Size	Release Date
GPT3.5	GPT (OpenAI)	gpt-3.5-turbo	175B	Mar 2023
Falcon	Falcon (TII) Finetuned by Nomic	gpt4all-falcon	7B	Jun 2023
Llama2	Llama (Meta)	Llama-2-7B	7B	Jul 2023
Gemini	Gemini (Google)	gemini-pro	Unknown	Dec 2023

the queries themselves, we construct two sets of queries with the aid of a template relating the parent and child entities, as illustrated in Figure 3.

Test Generation and Evaluation: We provide each LLM with each of the queries present in the two sets (i.e., original and mutated) in an atomic manner. We then feed the queries in a sequential manner with queries from both sets being presented in turn. This is then repeated with the order reversed. To invoke the oracles, we only consider responses that begin with a **yes** or **no**. Other responses are classified as "*Invalid*". This allows us to determine the accuracy of the responses without penalizing LLMs that fail to respond appropriately.

Knowledge Gap Mitigation: We evaluate the effectiveness of an ensemble method informed by KONTEST's test suite by determining the relative weights assigned to each SUT. We restrict the set of relations considered for this process such that a relation that yields an invalid answer in the query generation phase for any of the SUTs is not used in subsequent computations. We then perform five-fold cross validation on the remaining set of relations. Concretely, we hold out 20% of the relations for evaluation and use the consistency errors associated with the rest of the relations in the computation of the weights. We introduce a third template to evaluate whether our mitigation scheme generalizes to unseen templates.

Ablation Study: We assess whether an LLM could conceivably be used to replicate each of the subtasks in our approach by testing its applicability on the places domain. In particular, we chose to use `GPT3.5` for this due to its popularity and relatively high percentage (99.9%) of non-exempted responses in our experiments. In each case described below, we provide `GPT3.5` with sample questions and answers (few shot prompts) to ensure that the outputs adhere to the format of KONTEST.

To test the *knowledge construction capability* of `GPT3.5`, we ask `GPT3.5` where each of the 50 leaf nodes (used to evaluate KONTEST) is located. We then check the accuracy of the response and we exempted responses that were not in the requested format. To evaluate `GPT3.5`'s *test generation capability*, we provide the relation we are concerned with and ask it to generate two semantically equivalent queries involving the two entities. For ontological oracle, we provide `GPT3.5` with the full set of relations for each (knowledge) path and ask it to generate queries that expose inconsistencies in an LLM. For both oracles, if entity names and relations (as provided in the prompt) are not present in a generated query, then the test is considered invalid. For instance, "*Is there a County of Capellen in County of Capellen?*" is invalid since it does not correspond to the given knowledge relations. For *error detection*, we provide each set of queries along with their associated responses from each subject to `GPT3.5` and ask it to classify each set as being *Consistent* or *Inconsistent*. We also allow it to respond with *Invalid* when the SUT responses were exempted. For the ontological oracle, we provide `GPT3.5` with all the queries and responses relating to the entirety of a path and ask it to identify inconsistencies. We note that KONTEST performs its check in Algorithm 2 using the same set of inputs provided to `GPT3.5`.

Implementation Details and Platforms: KONTEST utilizes PyTorch 2.0, CUDA 11.3 and the llm package. All experiments were conducted on a GCP VM with an N1 series machine, eight vCPUs, 30 GB of memory and one Nvidia T4 GPU.

5 Evaluation Results

RQ1 Effectiveness: We found that KONTEST is effective in exposing consistency errors in LLMs: 19.2% of valid test executions result in consistency errors. Table 3 also shows that metamorphic errors (21.0% = 1784/8482) are more prevalent than ontological errors (8.9% = 133/1497). We attribute the lower error rate of ontological errors/inputs to the high complexity of our ontological oracle (*see* Table 1). Table 3 demonstrates that metamorphic errors are common across all LLMs. In particular, the tested LLMs are highly *inconsistent* when asked the same question in a different manner (*see* Table 1). In addition, we observe that a large model size does not necessarily lead to a smaller error rate than a smaller model, e.g., `GPT3.5` exhibits a metamorphic error rate of 27.2% for the places entities as compared to a metamorphic error rate of 17.1% for `Falcon`.

In addition, we validate the stability of our results by repeating our experiments four additional

Table 3: Effectiveness of KONTEST using a *knowledge graph*.

	LLM (Subject)	Valid Test Executions (%)						Errors (%)					
		Metamorphic				Ontological	All	Metamorphic				Ontological	All
		atomic	sequential -intra	sequential -inter	All Types			atomic	sequential -intra	sequential -inter	All Types		
Places	**Falcon**	254 (86.4)	530 (90.1)	538 (91.5)	1322 (89.9)	267 (91.4)	1589 (90.2)	50 (19.7)	24 (4.5)	152 (28.3)	226 (17.1)	25 (9.4)	251 (15.8)
	Gemini	294 (100)	588 (100)	588 (100)	1470 (100)	292 (100)	1762 (100)	51 (17.3)	6 (1.0)	104 (17.7)	161 (11.0)	40 (13.7)	201 (11.4)
	GPT3.5	293 (99.7)	588 (100)	587 (99.8)	1468 (99.9)	292 (100)	1760 (99.9)	61 (20.8)	201 (34.2)	138 (23.5)	400 (27.2)	17 (5.8)	417 (23.7)
	Llama2	266 (90.5)	543 (92.3)	542 (92.2)	1351 (91.9)	268 (91.8)	1619 (91.9)	94 (35.3)	52 (9.6)	209 (38.6)	355 (26.3)	27 (10.1)	382 (23.6)
	Total	1107 (94.1)	2249 (95.6)	2255 (95.9)	5611 (95.4)	1119 (95.8)	6730 (95.5)	256 (23.1)	283 (12.6)	603 (26.7)	1142 (20.4)	109 (9.7)	1251 (18.6)
Music	**Falcon**	141 (96.6)	287 (98.3)	287 (98.3)	715 (97.9)	94 (97.9)	809 (97.9)	24 (17.0)	38 (13.2)	69 (24.0)	131 (18.3)	3 (3.2)	134 (16.6)
	Gemini	146 (100)	292 (100)	292 (100)	730 (100)	96 (100)	826 (100)	39 (26.7)	8 (2.7)	77 (26.4)	124 (17.0)	7 (7.3)	131 (15.9)
	GPT3.5	146 (100)	292 (100)	292 (100)	730 (100)	96 (100)	826 (100)	40 (27.4)	85 (29.1)	56 (19.2)	181 (24.8)	5 (5.2)	186 (22.5)
	Llama2	137 (93.4)	283 (96.9)	276 (94.5)	696 (95.3)	92 (95.8)	788 (95.4)	46 (33.6)	63 (22.3)	97 (35.1)	206 (29.6)	9 (9.8)	215 (27.3)
	Total	570 (97.6)	1154 (98.8)	1147 (98.2)	2871 (98.3)	378 (98.4)	3249 (98.3)	149 (26.1)	194 (16.8)	299 (26.1)	642 (22.4)	24 (6.3)	666 (20.5)
	Total	1677 (95.3)	3403 (96.7)	3402 (96.6)	8482 (96.4)	1497 (96.5)	9979 (96.4)	405 (24.2)	477 (14.0)	902 (26.5)	1784 (21.0)	133 (8.9)	1917 (19.2)

times. We find that the open-source models with frozen weights (`Falcon` and `Llama2`) yielded identical results when compared to the initial experiments. However, the closed-source models exhibited (up to 6%) lower error rates on the subsequent iterations with `Gemini` exhibiting a less than one percent change in error rate. We attribute this to changes in the underlying models (OpenAI) as the additional experiments were performed approximately eight (8) months after the initial experiments. Furthermore, we find that the error rates for all four subsequent iterations are fairly similar indicating that the results are consistent when repeated within a short duration of time (within a few days). In particular, we find that `Gemini` results do not vary at all. Lastly, we also examine the stability of our results by executing both atomic and sequential queries separately. For instance, consider Q1 and Q6a (Table 1) and the following additional oracle: LLM(Q1)=LLM(Q6a). We found that only GPT3.5 exhibited inconsistent results for this oracle and further noted that the absolute number of errors found was negligible (<1%). These experiments demonstrate the stability of our results.

KONTEST *effectively exposes consistency errors in LLMS: 19.2% of valid test executions exposed a metamorphic or ontological error in LLMs.*

RQ2 Knowledge Coverage: Table 4 shows that *the tested LLMs cover about four-fifth (83.5%) of the tested knowledge graph across both templates.* We found that LLMs are particularly sensitive to the query templates. For instance, `Llama2` exhibits a 47.3% gap in knowledge for the places entities for one template, but only exhibits a 12.9% knowledge gap for the other. This suggests that LLMs may respond differently to two different, but semantically equivalent, input queries. We also observed

Table 4: Knowledge Coverage and Gap in tested LLMs (Highest coverage or gap are marked in **bold text)**

	LLM (Subject)	Relations	Knowledge Gap (%) Template 1	Template 2	Intersection	Overall Coverage (%)
Places	**Falcon**	294	12 (4.1)	93 (31.6)	9 (3.1)	**285 (96.9)**
	Gemini	294	60 (20.4)	71 (24.1)	40 (13.6)	254 (86.4)
	GPT3.5	294	**129 (43.9)**	107 (36.4)	**87 (29.6)**	207 (70.4)
	Llama2	294	38 (12.9)	**139 (47.3)**	37 (12.6)	257 (87.4)
Music	**Falcon**	146	20 (13.7)	15 (10.3)	3 (2.1)	**143 (97.9)**
	Gemini	146	60 (41.1)	45 (30.8)	33 (22.6)	113 (77.4)
	GPT3.5	146	63 (43.2)	27 (18.5)	25 (17.1)	121 (82.9)
	Llama2	146	**76 (52.1)**	**84 (57.5)**	**56 (38.4)**	90 (61.6)
	Total	1760	458 (26.0)	581 (33.0)	290 (16.5)	1470 (83.5)

that the smallest model (`Falcon`) has the lowest knowledge gap for the places entities (3.1%), while one of the biggest models (`GPT3.5`) has the highest knowledge gap (29.6%). This implies that model size/complexity is not a good proxy for knowledge coverage/gap. We do, however, attribute the performance of `Falcon` to its tendency to answer with a positive response regardless of the query. In addition, we also note that all queries posed to the subject LLMs are queries relating to an existing relation. These results show that KONTEST effectively reveals the knowledge gap in LLMs. We posit that knowledge coverage is a good criteria for estimating the underlying knowledge of an LLM.

KONTEST *effectively reveals knowledge gaps in LLMs: It exposes an average knowledge gap of 16.5% in the tested LLMs.*

RQ3 Knowledge Gap Mitigation: Results show that our *proposed mitigation technique reduces the SUT's knowledge gap by up to 39.30%.* Table 5 shows that our technique reduces the knowledge gap for all templates by 32.48%. We found that the simple majority voting ensemble worsens the SUTs' knowledge gap by up to 23.74%. We also observed that the mitigation performance of our technique generalizes to an unseen template (template three (3)). While the performance of our

Table 5: Knowledge Gap Mitigation results for KONTEST's weighted ensemble ("KONTEST"), versus simple majority voting ensemble ("Majority") and the initial *average* knowledge gap found in the SUTs ("SUTs"). Best reduction in knowledge gaps are in **bold text**. "%Impr. " means percentage improvement over the SUT.

Domain	Relations	Knowledge Gap (%)											
		Template 1			Template 2			Template 3			All (%Impr.)		
		KONTEST	Majority	SUTs	KONTEST	Majority	SUTs	KONTEST	Majority	SUTs	KONTEST	Majority	SUTs
Places	224	**28 (12.5)**	38 (17.0)	38.25 (17.1)	**49 (21.9)**	77 (34.4)	65.75 (29.4)	**30 (13.4)**	49 (21.9)	41.5 (18.5)	**107 (26.46)**	164 (-12.71)	145.5
Music	132	**39 (29.5)**	60 (45.5)	48.5 (36.7)	**15 (11.4)**	44 (33.3)	35.75 (27.1)	**24 (18.2)**	55 (41.7)	44.25 (33.5)	**78 (39.30)**	159 (-23.74)	128.5
Total	356	**67 (18.8)**	98 (27.5)	86.75 (24.4)	**64 (18.0)**	121 (34.0)	101.5 (28.5)	**54 (15.2)**	104 (29.2)	85.75 (24.1)	**185 (32.48)**	323 (-17.88)	274

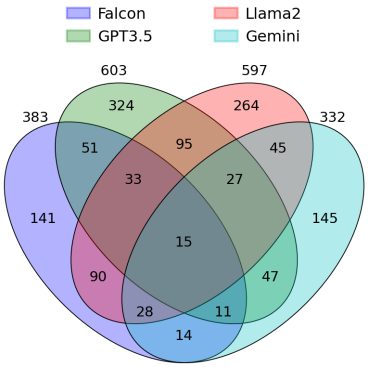


Figure 4: Venn Diagram of the consistency errors for each SUT.

Table 6: GPT3.5 efficacy in knowledge construction

	Total	Number of LLM Responses (%)				
		No Response	Exempted	Unexempted	Correct	Incorrect
Nodes	50	0 (0)	5 (10.0)	45 (90.0)	30 (60.0)	15 (30.0)
Edges	294	39 (13.3)	30 (10.2)	225 (76.5)	200 (68.0)	25 (8.5)

Table 7: Test generation effectiveness of GPT3.5 and KONTEST on the places entities. (**Meta.**: Metamorphic, **Onto.**: Ontological)

	LLM (Subject)	Valid Test Executions (%)			Errors (%)		
		Meta.	Onto.	All	Meta.	Onto.	All
GPT3.5	**Falcon**	1322 (89.9)	263 (93.9)	1585 (90.6)	232 (17.5)	25 (9.5)	257 (16.2)
	Gemini	1470 (100)	280 (100)	1750 (100)	172 (11.7)	41 (14.6)	213 (12.2)
	Llama2	1351 (91.9))	264 (94.3)	1615 (92.3)	370 (27.4)	27 (10.2)	397 (24.6)
	Total	4143 (93.9)	807 (96.1)	4950 (94.3)	774 (18.7)	93 (11.5)	867 (17.5)
	KONTEST w/o GPT3.5	4143 (93.9)	827 (94.4)	4970 (94.0)	742 (17.9)	92 (11.1)	834 (16.8)

mitigation technique is slightly better on the new template than the original templates the simple majority performs worse on an unseen template. On the whole, we find that our weighted ensemble is 42.72% ((323-185)/323) more effective than simple majority voting at reducing the knowledge gap. These results demonstrate the generalizability of our proposed mitigation technique and the efficacy of our weighted ensemble. We attribute the performance of our technique to the likelihood and distribution of inconsistencies discovered by KONTEST (Figure 4). Figure 4 shows that only 1.1% (15 out of 1330) of errors are found in all four SUTs while 65.7% (874 out of 1330) of errors are unique to a single SUT. This suggests that an ensembling scheme, informed by the relative performance of the SUTs, reduces knowledge gap.

The proposed ensemble method effectively reduces the SUT's knowledge gap by 32.48%.

RQ4 Ablation study with GPT3.5: We conduct an ablation study to investigate whether a state-of-the-art LLM (GPT3.5) is as effective as KONTEST in performing its three main sub-tasks – knowledge construction, test generation and error detection.

Knowledge Construction: Table 6 demonstrates that GPT3.5 is not a reliable knowledge constructor. GPT3.5 is only able to identify 68% of the relations present in the original knowledge graph. We believe this is because LLMs are generally intended to be a conversational engine, rather than a knowledge database (Pan et al., 2023). These results show the importance of knowledge graphs in KONTEST and suggest that LLMs are not a reliable replacement for knowledge graphs.

GPT3.5 *is an ineffective surrogate for a knowledge graph: It constructs only 68% of the ground-truth entity relations (edges).*

Test Generation: Given a few (three) query examples and the relations in a knowledge graph, GPT3.5 is slightly more effective (17.5% vs 16.8%) than KONTEST in test generation (see Table 7). We also observe that this effectiveness persists for both the metamorphic (18.7% vs 17.9%) and the ontological oracle (11.5% vs 11.1%). We attribute the performance of GPT3.5 to the effectiveness of few-shot prompting using the knowledge graph. This guides GPT3.5 to create tests similar to KONTEST.

GPT3.5 *is slightly (4.2%) more effective than* KONTEST *in generating tests, when provided entity relations with few-shot prompting.*

Error Detection: Table 8 highlights that while

Table 8: Error Detection Effectiveness (**TP/FP**: True/False positive, **UNK**: Unknown)

Subject	Metamorphic Errors (%)					Ontological Errors (%)		
	Kon-Test	GPT3.5			Paths	Kon-Test	GPT3.5	
		TP	FP	UNK			TP	FP
Falcon	226	208 (27.7)	453 (60.4)	89 (11.9)	100	23	23 (33.3)	46 (66.7)
Gemini	161	161 (97.6)	4 (2.4)	0 (0)	100	30	30 (54.5)	25 (45.5)
Llama2	355	352 (77.0)	93 (20.4)	12 (2.6)	100	22	20 (26.0)	57 (74.0)
Total	742	721 (52.6)	550 (40.1)	101 (7.4)	300	75	73 (36.3)	128 (63.7)

52.6% of the metamorphic errors detected by `GPT3.5` were detected correctly, approximately 40.1% of the errors were false positives [2]. For ontological error detection, we provided `GPT3.5` all possible queries and corresponding responses for each knowledge path. `GPT3.5` was then asked to detect any inconsistency in these responses. Unlike KONTEST, `GPT3.5` is unable to detect multiple inconsistencies in a knowledge path. Hence, for a fair comparison, we count ontological errors for KONTEST at the granularity of a knowledge path (i.e., at most one error per knowledge path). Since validating `GPT3.5` responses is not straightforward in this case, we manually validated the responses. We observed that the number of real ontological errors detected by `GPT3.5` is comparable to the ontological errors detected by KONTEST, but it also had a high false positive rate with nearly 63.7% of errors being misclassified.

> `GPT3.5` *detects both metamorphic and ontological errors, but exhibits a false positive rate of up to 63.7%.*

6 Related Work

LLMs and Knowledge: Pan et al. (Pan et al., 2023) presents techniques to combine LLMs and knowledge graphs to address their individual limitations. As an example, Huang et al. (Huang et al., 2023c) have demonstrated the feasibility of knowledge transfer to improve LLM's generalization ability in software engineering tasks. Analogously, WEAVER (Yang et al., 2023a) uses LLMs to generate knowledge bases, using which, requirements are extracted for testing models in real-world settings. GPT4GEO (Roberts et al., 2023) experimentally evaluates the capabilities and limitations of GPT-4 in geospatial domain (e.g., *places* entity), highlighting potential usage of GPT-4 in navigation. Unlike these proposed techniques, KONTEST employs knowledge graphs to expose inconsistencies and measure knowledge gaps in LLMs.

Testing and Analysis of LLMs: Several researchers have surveyed the challenges and opportunities for testing and analysing LLMs (Zhao et al., 2023; Hou et al., 2023; Zheng et al., 2023; Aleti, 2023). Researchers have also studied or proposed methods for testing and analyzing several quality properties of LLMs, including their reasoning ability (Wu et al., 2023; Qiu et al., 2023), non-determinism (Ouyang et al., 2023), interpretability (Palacio et al., 2023; Rodriguez-Cardenas et al., 2023), robustness (Zhu et al., 2023), fairness (Huang et al., 2023a), security concerns (e.g., privacy, memorization and backdoor attack) (Yang et al., 2023b; Staab et al., 2023; Huang et al., 2023b). In contrast to the aforementioned works, KONTEST studies the consistency and knowledge coverage of LLMs.

7 Conclusion

In this paper, we propose KONTEST where the key intuition is to distill (a subset of) facts from a knowledge graph, which are subsequently used to generate queries and formulate test cases for detecting a variety of consistency errors. Our evaluation reveals realistic consistency errors across state-of-the-art LLMs. Moreover, KONTEST opens opportunities to investigate LLMs through the lens of their knowledge gaps, which, in turn, is indicated as part of our framework. This helps designers and end users to understand and mitigate the effect of such knowledge gaps e.g., via prompt engineering or fine tuning. In future, we aim to investigate automated mitigation of consistency errors by leveraging the KONTEST framework. We provide our code and experimental data in the following:

`https://github.com/sparkssss/KonTest`

8 Limitations and Threats to Validity

Construct Validity: This relates to the metrics and measures employed in our experimental analysis. To mitigate this threat, we have employed standard testing metrics such as the number/rate of generated inputs, error-inducing inputs, (knowledge) coverage and testing time. For automatic analysis of hundreds of responses, our analysis does not handle expressive, non-binary LLM responses. However, we mitigate this by employing system prompts to ensure the model provides binary responses and we discard non-binary responses (as invalid).

[2] Note that some errors detected by `GPT3.5` cannot be automatically validated by our oracles (**UNK** in Table 8), as it excludes responses that do not begin with **yes/no**.

Internal Validity: This refers to the threat that our implementation of KONTEST performs its intended knowledge graph-based test generation. We conduct several manual and automated tests, as well as inspection of sampled outcomes of KONTEST to ensure the correctness of our approach. Our ablation study (**RQ4**) further allows us to probe the correctness of the sub-step of KONTEST versus using `GPT3.5`. We experimentally verify that over 90% of the entities we evaluate against existed in Wikidata prior to 2020. We also find that under 15% of the errors found by KONTEST are linked to these entities. In addition, we also note that the SUT ought to answer the question in a consistent manner regardless of whether the entity existed prior to the corresponding knowledge cutoff date.

External Validity: The main threat to external validity of this work is the generalizability of KONTEST and findings to LLMs, knowledge graphs, templates and entities beyond the ones used in this work. We mitigate the LLM generalizability threat by employing state-of-the-art off-the-shelf, mature, open model weights LLMs (`LLaMA2` and `FALCON`), as well as commercial LLMs (such as `GPT3.5` and `Gemini`). Similarly, our entity selection and template construction may be limited to our experimental setting. However, we demonstrate the applicability of KONTEST by using two different domains with multiple relations and entity types. However, we note that KONTEST might not be easily adapted to information that cannot be encapsulated within a knowledge graph. In addition, we employ few-shot prompting to conduct our ablation study, and our findings might not generalize to other prompting techniques. Finally, KONTEST employs Wikidata, a well-maintained knowledge graph that is popularly used in both academia and industry (Peters et al., 2019).

LLM Stability and Correctness: Researchers have identified several stability concerns about LLMs (Ouyang et al., 2023; Fan et al., 2023), such as non-determinism, randomness, sensitivity to prompting methods, and API/model updates. To mitigate these threats we performed several actions: First, we set the temperature of all models to zero (0), when possible (Cloud, 2023; Ouyang et al., 2023). Secondly, we reduce the risk of model updates by limiting our testing time (to about a day each per model) and checking for news of model updates before and after testing. Thirdly, we also use models with frozen weights (`Falcon` and `Llama2`) to reduce the non-determinism. To automatically validate model outcomes, we employ few-shot prompting, which has been shown to be effective for querying LLMs (Deng et al., 2023). We examined whether LLMs are comparable to KONTEST (**RQ4**) using `GPT3.5` since it produces the most valid responses (99.9%) (*see* Table 3).

Knowledge Graph Completeness and Soundness: We note that the knowledge graph is an incomplete snapshot of the real-world. In our evaluation, KONTEST only tests for facts (positive tests) derived from the knowledge graph. While it is also possible to use KONTEST to test for incorrect relation (negative test), the validation of such test results is challenging due to the incompleteness of the knowledge graph. Moreover, we only test 50 paths from this graph. However, these concerns do not affect our findings, as we are certain about the errors found within the tested subset of the knowledge graph. Finally, knowledge of the world naturally evolves over time and the knowledge graphs do not evolve at the same pace (Pan et al., 2023). To mitigate this, we use a widely used knowledge graph (Vrandečić and Krötzsch, 2014).

9 Ethics Statement

We elucidate our ethics statement in this section: (1) **Dataset**: We utilize data from Wikidata, a publicly available open knowledge base related to Wikipedia. Wikidata is licensed under the Creative Commons CC0 License. (2) **Human Evaluations**: Our experiments do not involve human participants. (3) **Approach**: We test KONTEST with the aid of LLMs (both proprietary and free). We acknowledge that these models may give biased results due to their training data and methods. However, we restrict our queries to existing relations in the knowledge graph making it unlikely to raise ethical concerns. We limit ourselves to running inference on pre-trained models due to the numerous environmental concerns (energy and water expenditure) associated with training these LLMs.

References

Aldeida Aleti. 2023. Software testing of generative ai systems: Challenges and opportunities. *arXiv preprint arXiv:2309.03554*.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2023. The reversal curse: Llms trained on "a is b" fail to learn "b is a".

Chart2000. Top 200 artists of 2000 to 2023. https://chart2000.com/artists.htm.

Jiaqi Chen, Bingqian Lin, Ran Xu, Zhenhua Chai, Xiaodan Liang, and Kwan-Yee K Wong. 2024. Mapgpt: Map-guided prompting for unified vision-and-language navigation. *arXiv preprint arXiv:2401.07314*.

Google Cloud. 2023. Vertex ai api.

Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 423–435.

Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. *arXiv preprint arXiv:2310.03533*.

Shahin Honarvar, Mark van der Wilk, and Alastair Donaldson. 2023. Turbulence: Systematically and automatically testing instruction-tuned large language models for code. *arXiv preprint arXiv:2312.14856*.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*.

Dong Huang, Qingwen Bu, Jie Zhang, Xiaofei Xie, Junjie Chen, and Heming Cui. 2023a. Bias assessment and mitigation in llm-based code generation. *arXiv preprint arXiv:2309.14345*.

Hai Huang, Zhengyu Zhao, Michael Backes, Yun Shen, and Yang Zhang. 2023b. Composite backdoor attacks against large language models. *arXiv preprint arXiv:2310.07676*.

Qing Huang, Yishun Wu, Zhenchang Xing, He Jiang, Yu Cheng, and Huan Jin. 2023c. Adaptive intellect unleashed: The feasibility of knowledge transfer in large language models. *arXiv preprint arXiv:2308.04788*.

IMF. Gdp per capita, current prices (u.s. dollars per capita). http://imf.org/external/datamapper/NGDPDPC@WEO/OEMDC/ADVEC/WEOWORLD?year=2023/.

Zhenlong Li and Huan Ning. 2023. Autonomous gis: the next-generation ai-powered gis. *arXiv preprint arXiv:2305.06453*.

Mapbox. Mapgpt: Deliver natural conversations with a location-intelligent ai assistant. https://www.mapbox.com/mapgpt.

Marcus J Min, Yangruibo Ding, Luca Buratti, Saurabh Pujar, Gail Kaiser, Suman Jana, and Baishakhi Ray. 2023. Beyond accuracy: Evaluating self-consistency of code large language models with identitychain. *arXiv preprint arXiv:2310.14053*.

Nomic AI. 2023. Falcon 7b model.

OpenAI. Openai deprecation policy. https://platform.openai.com/docs/deprecations/incremental-model-updates.

OpenAI. 2023. Gpt3.5 models.

Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. 2023. Llm is like a box of chocolates: the non-determinism of chatgpt in code generation. *arXiv preprint arXiv:2308.02828*.

David N Palacio, Alejandro Velasco, Daniel Rodriguez-Cardenas, Kevin Moran, and Denys Poshyvanyk. 2023. Evaluating and explaining large language models for code using syntactic structures. *arXiv preprint arXiv:2308.03873*.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2023. Unifying large language models and knowledge graphs: A roadmap. *arXiv preprint arXiv:2306.08302*.

Matthew E Peters, Mark Neumann, Robert L Logan IV, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. 2019. Knowledge enhanced contextual word representations. *arXiv preprint arXiv:1909.04164*.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, et al. 2023. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. *arXiv preprint arXiv:2310.08559*.

Jonathan Roberts, Timo Lüddecke, Sowmen Das, Kai Han, and Samuel Albanie. 2023. GPT4GEO: how a language model sees the world's geography.

Daniel Rodriguez-Cardenas, David N Palacio, Dipin Khati, Henry Burke, and Denys Poshyvanyk. 2023. Benchmarking causal study to interpret large language models for source code. *arXiv preprint arXiv:2308.12415*.

June Sallou, Thomas Durieux, and Annibale Panichella. 2023. Breaking the silence: the threats of using llms in software engineering. *arXiv preprint arXiv:2312.08055*.

Robin Staab, Mark Vero, Mislav Balunović, and Martin Vechev. 2023. Beyond memorization: Violating privacy via inference with large language models. *arXiv preprint arXiv:2310.07298*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2023. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. *arXiv preprint arXiv:2307.02477*.

Chenyang Yang, Rishabh Rustogi, Rachel Brower-Sinning, Grace A Lewis, Christian Kästner, and Tongshuang Wu. 2023a. Beyond testers' biases: Guiding model testing with knowledge bases using llms. *arXiv preprint arXiv:2310.09668*.

Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, DongGyun Han, and David Lo. 2023b. What do code models memorize? an empirical study on large language models of code. *arXiv preprint arXiv:2308.09932*.

Bangguo Yu, Hamidreza Kasaei, and Ming Cao. 2023. L3mvn: Leveraging large language models for visual target navigation. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Zibin Zheng, Kaiwen Ning, Jiachi Chen, Yanlin Wang, Wenqing Chen, Lianghong Guo, and Weicheng Wang. 2023. Towards an understanding of large language models in software engineering tasks. *arXiv preprint arXiv:2308.11396*.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Neil Zhenqiang Gong, Yue Zhang, et al. 2023. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv preprint arXiv:2306.04528*.